Model!T

User guide

Modelit XML Toolbox for Matlab

Manual:	Modelit XML Toolbox for Matlab
	User guide
Authors:	Kees-Jan Hoogland
	Nanne van der Zijpp
Copyright:	2006-2010, Modelit
Contact:	info@modelit.nl
	www.modelit.nl

Contents

iv

1   Introduction

1.1   Matlab and XML

XML (E**X**tensible **M**arkup **L**anguage) is designed to store complex datastructures in plain text format to provide a software- and hardware-independent way of structuring, storing and sharing data. Matlab R2006a offers some routines for processing XML documents but using these functions requires knowledge of Java and the Document Object Model (DOM).

1.2   Modelit XML toolbox for Matlab

The Modelit XML toolbox for Matlab aims to make XML functionality available to Matlab users without requiring extensive knowledge of Java or the Document Object Model. Although it provides easy access to many XML features, experienced users can still use their knowledge of Java and the Document Object Model to their advantage, as the toolbox supports features like XPath as well.

Where possible the Modelit XML toolbox for Matlab exploits the analogy between Matlab structures and XML documents to provide an intuitive access to XML data and provides the beginning user with the following features:

- Import and export data to XML format;
- Accessing XML data from the command prompt and Matlab m-files;
- Conversion of XML data to matlab structures;
- Visualizing the XML tree structure.

For the more demanding users, some advanced features are supported:

- Handling namespaces and attributes;
- Validating with DTD or XSD;
- Complete XPath syntax to extract information from XML documents;
- Transform XML documents to HTML.

1.3   Compatibility with Matlab versions

The Modelit XML toolbox for Matlab has been extensively tested with Matlab R2006b, but should work with nay Matlab version from 2006a and later. The XML toolbox can also be used together with the Matlab compiler.

1.4   Downloading a trial version of the Modelit XML Toolbox

A free version of the Modelit XML toolbox for Matlab is available from our website, www.modelit.nl. This version has all functionalities of the Modelit XML toolbox but does not contain the m-files. Although your feedback is appreciated we cannot guarantee support on the free version.

1.5   Buying the Modelit XML Toolbox

If you intend to use the XML toolbox for commercial applications, you may want to purchase a full license which entitles you to support and access to the source code.

A full license for the Modelit XML toolbox can be purchased at €350 and includes one year of support and updates. Please contact info@modelit.nl for more information.

2   Installation

Please follow the next steps to install automatically install the Modelit XML toolbox:
1.   Unzip the files from the XMLToolbox.zip file. This creates a folder 'xml_toolbox'

   We refer to the full path of this folder as XMLPATH.

2.   Add the following lines to your startup.m file, with:

```
%Define toolbox location (adapt if needed)
XMLPATH='c:\userfiles\xml_toolbox';

%add to matlab path:
path(XMLPATH,path);

%add to dynamic JAVA class path:
javaaddpath([XMLPATH filesep 'java' filesep 'jaxen-full.jar' ]);
javaaddpath([XMLPATH filesep 'java' filesep 'Saxpath.jar' ]);
javaaddpath([XMLPATH filesep 'java' filesep 'xmltoolbox.jar' ]);
```

3.   Restart Matlab.

The directory structure of the Modelit XML Toolbox for Matlab is shown in Figure 1 the following directories:

- ***root,***
  The files in this directory are described in section 5.3
- ***@xml (the XML-object)***
  The files in this directory are the methods of the XML-object are described in section 5.2
- ***@xml/private***
  The files in this directory are the private methods of the XML-object and are described in section 5.3.1
- ***examples***
  This directory contains example XML files, listed in section 6
- ***java***
  Directory with the necessary java archives:
  - ***Jaxen-full.jar***
    Contains the Jaxen API to the XPath engine, available from www.jaxen.org
  - ***Saxpath.jar***
    Contains java classes related to the event-based parsing and handling of XPath expressions, available from www.jaxen.org
  - ***xmltoolbox.jar***
    Contains java classes for visualizing XML DOM trees and an implementation of an defaulterrorhandler for XML validation.
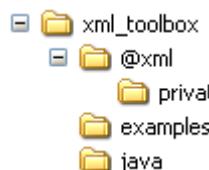
Figure 1: *Directory structure of the XML toolbox*

3

The size of the XML documents that can be handled by the Modelit XML toolbox is limited by the available memory on the heap space of the Java Virtual Machine in which Matlab is active. This heap size can be increased by creating a java.opts file in the $MATLAB/bin/$ARCH ($MATLAB is the root directory and $ARCH is your system architecture), or in the current directory when you start Matlab, containing the following command:

-Xmx268435456

This will make 256MB of JVM memory available, the parameter can be adjusted as needed. How to increase the Java Virtual Machine heap space is also explained in:

http://www.mathworks.com/support/solutions/data/1-18I2C.html?solution=1-18I2C

Listing 1: *java.opts with heap size of 256 megabytes*

```
-Xmx26435456
```

3   Getting started with the Modelit XML toolbox for Matlab

3.1   Introduction

Figure 2 illustrates the high-level functions in the XML-toolbox and how they can be used to convert between:
- XML documents - files that contain XML code;
- XML strings - Matlab character arrays that contain XML code;
- Matlab data structures - "Normal" Matlab variables;
- Matlab XML objects - a Matlab object created by the XML toolbox.

In many cases, these high-level functions will be all you need to access or store data in XML-format, and there is no need to read beyond this chapter. However, if you do, you will discover that the XML toolbox has many additional useful additional functions.

3.2   A note on the analogy between XML and Matlab data structures

The Modelit XML toolbox can map any Matlab data structure to an XML structure. The opposite is not true. XML allows datasets that can't be converted to a Matlab structure by the Modelit XML toolbox. Typical constructs that cannot be converted are:
- attributes that are specified with a fields;
- concatenations of dissimilar fields or structures.

The command "S=xml2struct(xml(XMLFILE))" converts an XML dataset to an equivalent Matlab structure S and you can access all data using only the variable S.

In many cases the XML file will contain a dataset that cannot be mapped to a Matlab structure. In these case you will see the message:

```
>> xml2struct(xml('test.xml'))
??? Error using ==> xml.xml2struct
XML contents do not fit in Matlab structure
```

 In these cases you can still access the XML data, but you will need to do this using the XML object:
```
>> s=xml('test.xml')
>> disp(s.fld)
```
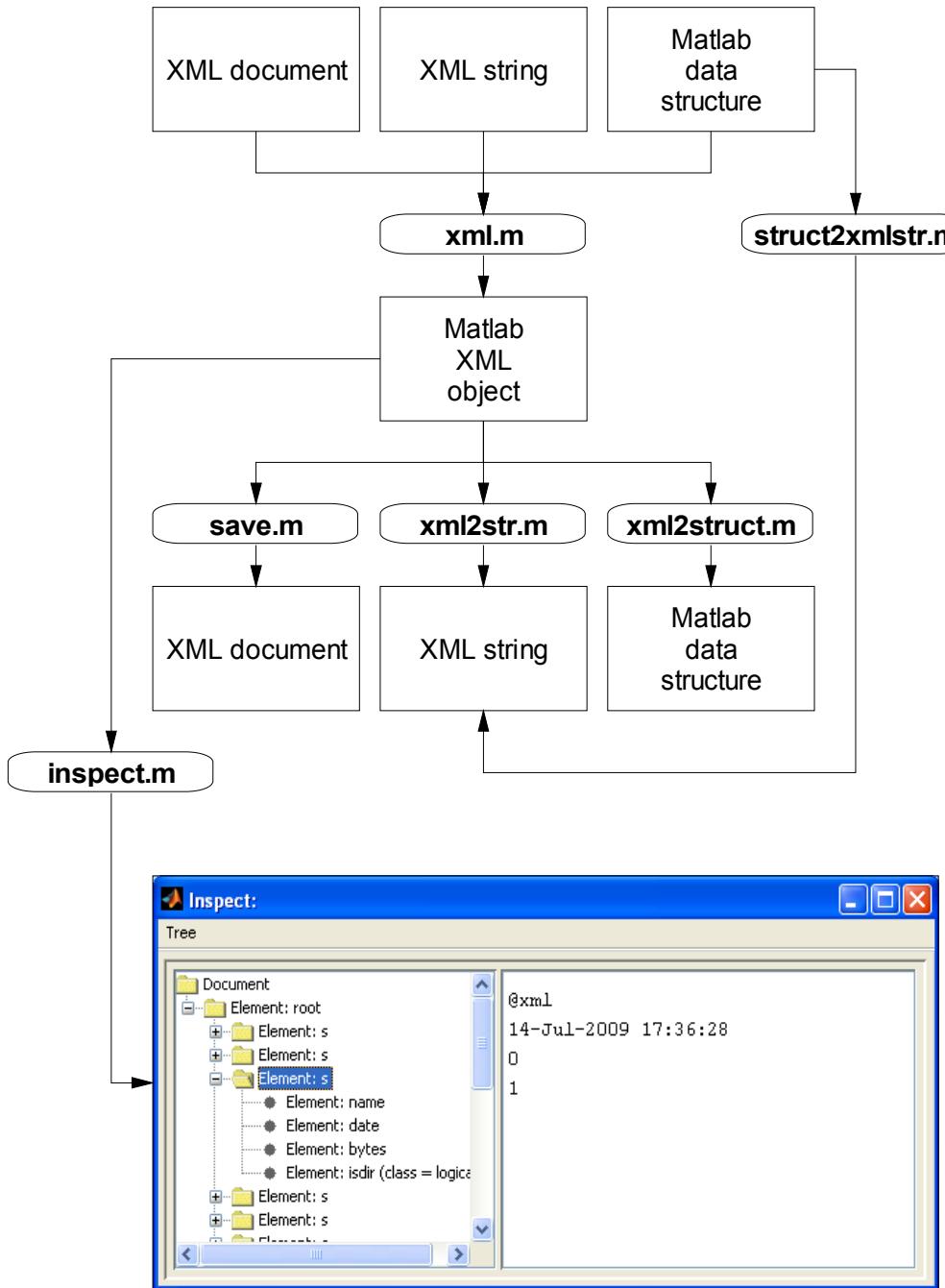
Figure 2: *Overview of the most popular uses of the toolbox*

3.3 xml2str and struct2xmlstr

As shown in Figure 2, there are 2 routes to convert a Matlab structure to an XML string:
- The first way is to convert the Matlab structure to an XML object and then to invoke the method "xml2str". The equivalent command is:
    str=xm2str(xml(S))
    The function xml2str relies on Java methods to map XML data to ASCII text.

- The second way is to use the function "struct2xmlstr". This function skips creation of an XML object and does the conversion directly. The function struct2xmlstr is designed primarily to be as fast as possible, and does not rely on any Java method.

The commands xm2str(xml()) and struct2xmlstr() are similar but not fully equivalent. The main differences are:
- struct2xmlstr does not save the size of numeric arrays. xml2str saves the array size as an attribute.
- struct2xmlstr looks at the datatype to decide if a specific numeric value is saved with or without digits. xml2str looks at the actual value.

See section 3.4 for examples of Matlab variables converted to XML strings by xml2str and struct2xmlstr.

3.4 Saving Matlab data as XML: some examples

The examples on the next page illustrate how the high level routines in the XML toolbox convert Matlab structures to XML code. The table is also used to illustrate the differences between the quick-and-dirty routine "struct2xmlstr" and the "official" implementation "xml2str".

Matlab code	Resulting XML after xml2str(xml(s))	Resulting XML after struct2xmlstr (s)
s.fld='stringvalue'	`<?xml version="1.0" encoding="ISO-8859-1"?>` `<root><fld>stringvalue</fld></root>`	

data(1,1).fld='stringvalue1'
data(1,2).fld='stringvalue2'
s.data=data
 	
data(1,1).fld='stringvalue1'
data(2,1).fld='stringvalue2'
s.data=data
	`<?xml version="1.0" encoding="ISO-8859-1"?>`
	`<root>`
	`<data><fld>stringvalue1</fld></data>`
	`<data><fld>stringvalue2</fld></data>`
	`</root>`
	
	Note: array sizes of struct arrays are not stored in XML files	

s.fld={'aa','bb'};	`<?xml version="1.0" encoding="ISO-8859-1"?>`
	`<root>`
	`<fld>aa</fld>`
	`<fld>bb</fld>`
	`</root>`	

s.fld=strvcat('aa','bb');	`<?xml version="1.0" encoding="ISO-8859-1"?>`
	`<root>`
	`<fld>aa`
	`bb`
	`</fld>`
	`</root>`
	
	Note: multiline text arrays are stored line by line between a single pair of tags. Newline characters are included to separate lines.
		`<?xml version="1.0" encoding="ISO-8859-1"?>`
		`<root>`
		`<fld>abab</fld>`
		`</root>`
		
		Note: multiline text arrays may not be stored as intended. You may want to convert the text arrays to cell arrays first to force correct behavior.

s(1).fld='stringvalue1'
s(2).fld='stringvalue2'
	-Not possible-
	
	Note: This option is not implemented because a valid XML document can only have 1 root.
		`<?xml version="1.0" encoding="ISO-8859-1"?>`
		`<root><fld>stringvalue1</fld></root>`
		`<root><fld>stringvalue2</fld></root>`
		
		Note: struct2xmlstr ignores the fact that above structure has no unique root.

s.fld=
12345678901234567890
	`<?xml version="1.0" encoding="ISO-8859-1"?>`
	`<root>`
	`<fld> 1234567890123456800</fld>`
	`</root>`
	
	Note: xml2str prints numeric values with single or double precision with the %f or %.0f format specifier, depending on the actual value. Integer valued data are stored without digits, float valued data are stored with digits. In both cases a maximum of 17 significant numbers is printed.
		`<?xml version="1.0" encoding="ISO-8859-1"?>`
		`<root>`
		`<fld> 1234567890123456800.000000</fld>`
		`</root>`
		
		note: struct2xmlstr prints numeric values with single or double precision with the %f format specifier, regardless of the actual value. This implies 6 digits and a maximum of 17 significant numbers.

s.fld=1:4	`<?xml version="1.0" encoding="ISO-8859-1"?>`
	`<root>`
	`<fld size="1  4">1  2  3 4</fld>`
	`</root>`
	
	Note: xml2str stores array sizes as an attribute.
		`<?xml version="1.0" encoding="ISO-8859-1"?>`
		`<root>`
		`<fld>1.000000 2.000000 3.000000 4.000000</fld>`
		`</root>`

s.fld=[1 2;3 4]	`<?xml version="1.0" encoding="ISO-8859-1"?>`
	`<root>`
	`<fld size="2  2">1  2`
	`3  4`
	`</fld>`
	`</root>`
	
	Note: xml2str exports a multiline numeric array line by
		`<?xml version="1.0" encoding="ISO-8859-1"?>`
		`<root>`
		`<fld>1.000000 3.000000 2.000000 4.000000</fld>`
		`</root>`
		
		Note: *struct2xmlstr only considers the number of*

	line. Newline characters are included to separate lines.	*elements in an array and ignores height and width attributes.*

s.fld=uint8(1:3)	`<?xml version="1.0" encoding="ISO-8859-1"?>`
	`<root>`
	`<fld>1 2 3</fld>`
	`</root>`	

s.fld=[1.1;pi]	`<?xml version="1.0" encoding="ISO-8859-1"?>`
	`<root><fld size="2  1">   1.1`
	`3.1416`
	`</fld></root>`
		`<?xml version="1.0" encoding="ISO-8859-1"?><root>`
		`<fld>1.100000 3.141593</fld></root>`

s.fld=[]	`<?xml version="1.0" encoding="ISO-8859-1"?>`
	`<root>`
	`<fld/>`
	`</root>`
	
	Note: all empty arrays are stored in the same way regardless of their type.	

s.fld=''		

4 Tutorial

If you only need to access and manipulate XML documents it is sufficient to read sections 4.1, 4.2, 4.3 and 4.4. The remaining sections describe more advanced or specific features of the Modelit XML toolbox.

4.1 The XML object

All functions in the Modelit XML toolbox are implemented through the XML object. This object contains the XML document in the form of a Java Document Object Model (DOM). The methods for accessing and manipulating the XML contents are such that from the outside world the XML object looks and acts like a normal Matlab structure.

An XML object can be created with the command xml which has the following syntax, see Listing 2 and section 5.1.1.

Listing 2: *Syntax of the XML-object constructor*

```
Obj = xml(FileName,isNameSpaceAware,isValidating)
```

The third input argument (isValidating) is discussed in section 4.5
The second input argument (isNamespaceWare) is discussed in section 4.8
The first input argument (FileName) can be one of the following types:

- *Empty*
  An empty XML object will be created, see Listing 3.

  Listing 3: *Construction of an XML object with no input arguments*

  ```
  >> obj = xml

     xml-object (root: root)
     number of nodes: 1
     <no fields available>
  ```

- *File*
  The easiest way to create an XML object is from an XML file, the input argument is then either a string with the location of the XML file or a java.io.File,
  see Listing 4.

  Listing 4: *Construction of an XML object from an XML file*

  ```
  >> obj = xml(fullfile(pwd,'examples','books.xml'))

     xml-object (root: bookstore)
     number of nodes: 25
     fieldnames:
     - book (4)

  >> obj = xml(java.io.File(fullfile(pwd,'examples','books.xml')))

     xml-object (root: bookstore)
     number of nodes: 25
     fieldnames:
     - book (4)
  ```

- *XML string*
  An XML object can be created directly from an XML string, see Listing 5.

Listing 5: *Construction of an XML object directly from an XML string*

```
>> str = '<book><title>Harry Potter</title><author>J.K.Rowling</author></book>'
>> obj = xml(str)

    xml-object (root: book)
    number of nodes: 3
    fieldnames:
    - author (1)
    - title  (1)
```

- *Matlab structure*
  a standard Matlab structure or struct array can be converted to an XML object.
  Matlab objects will first be converted to Matlab structures and then converted to
  an XML object, see Listing 6

Listing 6: *Construction of an XML object from a Matlab structure*

```
>> S = dir
>> obj = xml(S)

    xml-object (root: root)
    number of nodes: 41
    fieldnames:
    - bytes (10)
    - date  (10)
    - isdir (10)
    - name  (10)
```

- *Java inputstream*
  it is possible to create an XML object from an Java inputstream, this can be useful
  with for example web services, see Listing 7

Listing 7: *Creating an XML object from an java inputstream*

```
>> obj = xml(java.io.FileInputStream((fullfile(pwd,'examples','books.xml'))))

    xml-object (root: bookstore)
    number of nodes: 25
    fieldnames:
    - book (4)
```

- *Java DOM object*
  the core of the XML object is a DOM representation of an XML document, the
  XML object can therefore directly be constructed from a DOM object. This DOM
  object can also be constructed with the Matlab command xmlread, see Listing 8

Listing 8: *Construction of an XML object from a java DOM object*

```
>> DOM = xmlread(fullfile(pwd,'examples','books.xml'));
>> obj = xml(DOM)

    xml-object (root: bookstore)
    number of nodes: 25
    fieldnames:
    - book (4)
```

When an XML object is constructed the following information is displayed:

- ***The name of the root node***
  an XML document always has a unique root node, the name of the root node is sometimes necessary when using XPath, see section 4.9.
- ***Total number of nodes***
  of which the tree representation of the XML document consists.
- ***The names of the nodes***
  which appear directly under the root node plus the number of times they occur. These names can be used by the user to access and manipulate the XML content, see section 4.2.

Furthermore an XML object consists of three fields, these fields can be accessed and manipulated with the XML object get and set methods, see sections 5.2.20 and 5.2.21:

- ***DOM***
  the tree representation of the XML document.
- ***File***
  the name of the source file from which the XML object was created or the name of the file to which the XML object was saved to.
- ***NS***
  the namespaces, see section 4.8.

4.2   Accessing and manipulating data

To the outside world the XML-object appears to be a normal Matlab structure and accessing en manipulating data happens in the same way. With some minor differences:

1. ***The return value is a cell array***
   it is possible that the return values consist of different types, because Matlab cannot handle arrays with mixed element types the return values are wrapped in a cell array, see Listing 9 for an example in which two element of different type are returned.

Listing 9: *Returning of mixed element types from an XML object*

```
>> obj = xml(fullfile(pwd,'examples','mixed.xml'))
>> items = obj.item;
>> items{:}

    xml-object (root: root)
    number of nodes: 7
    fieldnames:
    - ARTIST  (1)
    - COMPANY (1)
    - COUNTRY (1)
    - PRICE   (1)
    - TITLE   (1)
    - YEAR    (1)

    xml-object (root: root)
    number of nodes: 5
    fieldnames:
    - author (1)
    - price  (1)
    - title  (1)
    - year   (1)
```

2. **_Invalid Matlab field names can be used in combination with XML objects_**
   by wrapping them in ('…'), e.g. the invalid Matlab field ('a-b')
3. **_multi-level indexing is possible_**
   statements such as 'books(3:4).author' are valid

Listing 10: *Accessing data in an XML object*

```
>> books = xml(fullfile(pwd,'examples','books.xml'))
>> books = obj.book

books =

    [1x1 xml]
    [1x1 xml]
    [1x1 xml]
    [1x1 xml]

>> books{1}

    xml-object (root: root)
    number of nodes: 5
    fieldnames:
    - author (1)
    - price  (1)
    - title  (1)
    - year   (1)
```

Besides retrieving contents from an XML object it is also possible to manipulate the XML contents in the same way as the data that is stored in a Matlab structure can be manipulated. If necessary extra fields will be added to the XML object, see for example Listing 11. The following types can be added to an XML object:

- **_XML object_**
  it is possible to add another XML object to a field in the XML object, the two XML objects will then be combined into one single XML object, see Listing 11.

Listing 11: *Add an XML object to an XML object*

```
>> books = xml(fullfile(pwd,'examples','books.xml'));

    xml-object (root: bookstore)
    number of nodes: 25
    fieldnames:
    - book (4)

>> book = xml(fullfile(pwd,'examples','book.xml'))

    xml-object (root: root)
    number of nodes: 6
    fieldnames:
    - author (2)
    - price  (1)
    - title  (1)
    - year   (1)

>> books.book(5) = book

    xml-object (root: bookstore)
    number of nodes: 31
    fieldnames:
    - book (5)
```

- ***String***
  it is possible to add a string or an array of strings to an XML object, see Listing 12.

Listing 12: *Add a string to an XML object*

```
>> obj = xml;
>> obj.string = 'string';
>> obj.array = strvcat('line 1','line 2')

    xml-object (root: root)
    number of nodes: 3
    fieldnames:
    - array  (1)
    - string (1)
```

- ***Number***
  A number or matrix can also be added to an XML object, if the matrix is sparse it will be converted to a full matrix, see Listing 13.

Listing 13: *Add a number of matrix to an XML object*

```
>> obj = xml;
>> obj.number = 123;
>> obj.matrix = rand(3);
>> obj.sparse = speye(3)

    xml-object (root: root)
    number of nodes: 4
    fieldnames:
    - matrix (1)
    - number (1)
    - sparse (1)
```

- ***Matlab structure***
  A Matlab structure or structarray will be converted to an XML object and then added to the XML object. Matlab objects will be converted to Matlab structures and then added to the XML object, Java objects will be converted to strings and then added to the XML objects.

Listing 14: *Adding structures to an XML object*

```
>> obj = xml;
>> obj.dir = dir;
>> obj.java = javax.swing.JButton

    xml-object (root: root)
    number of nodes: 43
    fieldnames:
    - dir  (1)
    - java (1)
    xml-object (root: root)
    number of nodes: 6
    fieldnames:
    - author (2)
    - price  (1)
    - title  (1)
    - year   (1)

>> books.book(5) = book

    xml-object (root: bookstore)
    number of nodes: 31
    fieldnames:
    - book (5)

>> inspect(books)
```

- **_Matlab cell array_**
  it is not possible to add a cell array to an XML object, just add the cell contents separately to the XML object.

With the function isfield (see section 5.2.7) it can be checked if a certain field exists in an XML document, similarly with the function rmfield (see section 5.2.9) fields can be removed from the XML document. The two functions are similar to the Matlab equivalents with the difference that the path to the field must be a string expression, see Listing 15.

Listing 15: *Removing a field from an XML object*

```
>> books = xml(fullfile(pwd,'examples','books.xml'))
>> isfield(books,'book(1).author')
ans =
     1
>> rmfield(books,'book(1).author');
>> isfield(books,'book(1).author')
ans =
     0
```

4.3   Viewing the XML content

The XML contents of an XML object can be viewed by the user in two different formats:

1. **_Plain ASCII_**
   the view method of the XML object displays the XML content on the console, see Listing 16 for an example.

Listing 16: *View XML contents.*

```
>> obj = xml(fullfile(pwd,'examples','book.xml'));
>> view(obj)

<?xml version="1.0" encoding="ISO-8859-1"?>
<book category="MATHS">
<title lang="en">Probability and Random Processes</title>
<author>G.R. Grimmett</author>
<author>D.R. Stirzaker</author>
<year>1992</year>
<price>39.00</price>
</book>
```

2. **_Tree structure_**
   the inspect method of the XML object displays the XML content in a separate window as a navigable tree structure, see Figure 3 for an example.

Listing 17: *Inspect XML contents.*

```
>> obj = xml(fullfile(pwd,'examples','books.xml'));
>> inspect(obj)
```
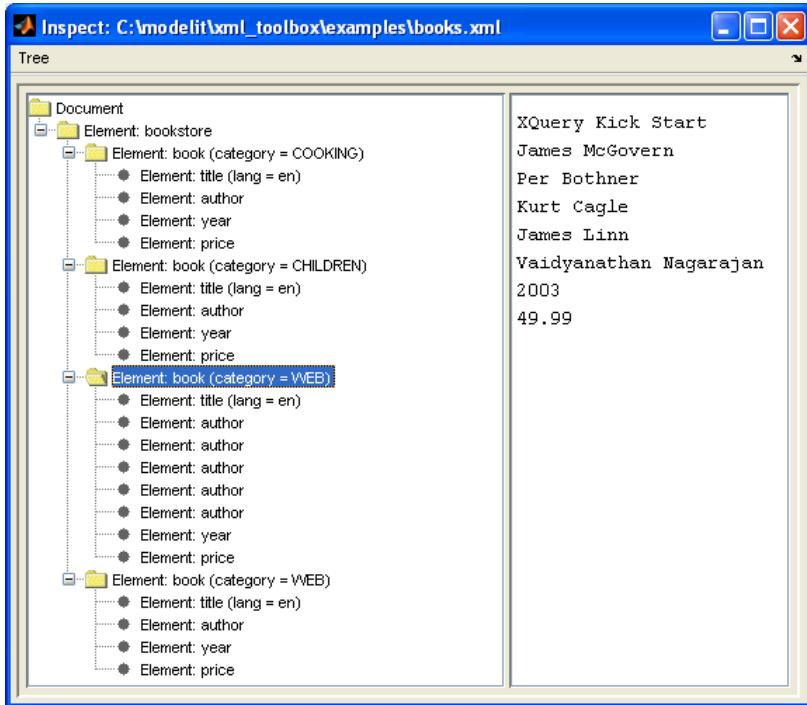
Figure 3: *Inspection of the tree structure of an XML document.*

Three actions can be selected In the 'Tree'-menu in the menu bar on the top of the XML structure inspector:

1. ***Collapse all***
   collapse the entire XML tree to one single node
2. ***Expand all***
   expand the entire XML tree, so all nodes are visible
3. ***Save selected node***
   save the selected node and its subnodes to an XML file
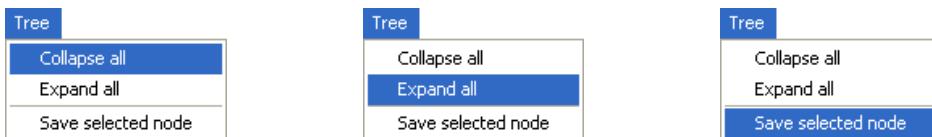
Figure 4: *Choices in the 'Tree'-menu in the XML structure inspector.*

4.4   Saving the XML content

The XML content of an XML object can be saved to an XML file by using the save method of the XML object, see Listing 18.

Listing 18: *Saving an XML object.*

```
>> obj = xml                %create an empty xml object
>> obj.date = datestr(now)    %add fields with values
>> obj.description = 'test'
>> obj = save(obj,'test.xml') %save object by specifying filename
```

A save dialog will appear if the second input argument of the save method is not specified, see Figure 5:
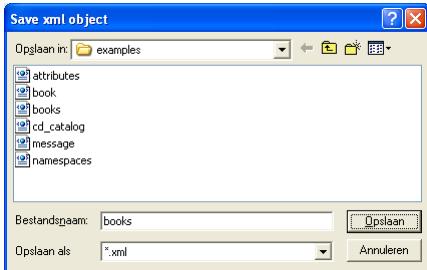
Figure 5: *Save dialog.*

The save method returns an updated XML object in which the name of the specified file is stored. The current filename can be retrieved with the get method of the XML object, see Listing 19.

Listing 19: *Retrieve the filename of an XML object.*

```
get(obj,'file')

ans =

c:\modelit\xml_toolbox\test.xml
```

4.5   Validating an XML document

XML requires the XML document be 'well formed' i.e. each XML document has to conform to the correct XML syntax, such as:

- All XML elements have a closing tag
- All XML elements are properly nested
- Attributes are quoted
- There is one root element

But because XML tags are not predefined, it might be necessary to check if an XML document has the right format. For example a format which can be used by a certain application. For this reason the XML object constructor can take three input arguments as described in section 4.1 and Listing 50 which makes it possible to validate XML document against a DTD (Document Type Definition) or XSD (XML Schema Definition):

> ***DTD***
> with a DTD, each XML document can carry a description of its own format with it which defines the structure with a list of legal elements. See Listing 20 and Listing 21 for an example of an XML document with a DTD.

> **Listing 20:** note_dtd.xml

```
<?xml version="1.0"?>
<!DOCTYPE note SYSTEM "note.dtd">
<note>
        <to>Tove</to>
        <from>Jani</from>
        <heading>Reminder</heading>
        <body>Don't forget me this weekend!</body>
</note>
```

Listing 21: *note.dtd*

```
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

The XML document of Listing 20 and Listing 21 can be parsed by the toolbox function `xml` as:

obj = xml(fullfile(pwd,'examples','note_dtd.xml'),0,1)

where the third argument (isValidating) is set to true. Normally the second argument (isNameSpaceAware) is set to false (DTD cannot handle namespaces). See http://www.w3schools.com/dtd for more information about DTD, and how to validate XML documents.

XSD (XML Schema Definition)
XML Schema is an XML based alternative to DTD and is more flexible, for instance it can handle namespaces and supports data types. Listing 22 and Listing 23 are an example of an XML document with an XSD.

.

Listing 22: *note_xsd.xml*

```
<?xml version="1.0"?><note
xmlns="http://www.w3schools.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.w3schools.com note.xsd">
        <to>Tove</to>
        <from>Jani</from>
        <heading>Reminder</heading>
        <body>Don't forget me this weekend!</body>
</note>
```

Listing 23: *note.xsd*

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.w3schools.com"
xmlns="http://www.w3schools.com"
elementFormDefault="qualified"><xs:element name="note">
    <xs:complexType>
      <xs:sequence>
       <xs:element name="to" type="xs:string"/>
       <xs:element name="from" type="xs:string"/>
       <xs:element name="heading" type="xs:string"/>
       <xs:element name="body" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
</xs:element></xs:schema>
```

The XML document of Listing 22 and Listing 23 can be parsed by the toolbox function `xml` as:

obj = xml(fullfile(pwd,'examples','note_xsd.xml'),1,1)

where the second argument (isNameSpaceAware) is set to true (XSD can handle namespaces) and the third argument (isValidating) also set to true.

See http://www.w3schools.com/schema for more information about XSD, and how to validate XML documents.

4.6   Transforming an XML document with a stylesheet

XML was created to store data. No information about how to display this data is included in the XML document itself. By separating the style from the content the same data can be presented in different ways.

XSLT (Extensible Stylesheet Language Transformations) can be used to display an XML document by transforming it to another XML document, or another type of document that is recognized by a browser, like HTML and XHTML. With XSLT elements and attributes can be added or removed from the output file, elements can be rearranged, tests can be performed and decisions can be made about which elements to hide and display. See http://www.w3schools.com/xsl/ for more information.

By using the xslt method of the XML-object an XML document can be transformed into an HTML document which can be displayed in a browser by using the Matlab command 'web'. See Listing 24 and Figure 6.

Listing 24: *Transforming XML to HTML.*

```
>> obj = xml(fullfile(pwd,'examples','cd_catalog.xml'))
>> S = xslt(obj,fullfile(pwd,'examples','cd_catalog.xsl'))

S =

<html>
   <body>
      <h2>My CD Collection</h2>
      <table border="1">
         <tr bgcolor="#9acd32">
            <th>Title</th>
            <th>Artist</th>
         </tr>
      </table>
   </body>
</html>

>> web(['text://' S],'-new', '-notoolbar');
```

Figure 6: *The cd_catalog.xml transformed to HTML*

4.7   Using attributes

XML elements can have attributes which can provide additional information about elements. While data is normally stored as elements, metadata (information about the data) is usually stored as attributes of elements. Listing 25 shows how attributes are added to the elements in an XML document.

Listing 25: *An XML document with attributes*

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<book category="MATHS">
        <title lang="en">Probability and Random Processes</title>
        <author>G.R. Grimmett</author>
        <author>D.R. Stirzaker</author>
        <year>1992</year>
        <price>39.00</price>
</book>
```

In the XML toolbox attributes can be accessed and manipulated in the same way as 'normal' elements, by using the symbol '@' to indicate that there is been referred to an attribute. As the '@' symbol is a reserved Matlab symbol it is necessary to put quotes and parentheses around it, see Listing 26.

Listing 26: *retrieving attribute values*

```
>> book = xml(fullfile(pwd,'examples','book.xml'));
>> category = book.('@category')

category =

    'MATHS'

>> lang = book.title.('@lang')

lang =

    'en'

>> book.title.('@lang') = 'es';
>> lang = book.title.('@lang')

lang =

    'es'
```

Attributes can be removed from the XML object in the same way as described in 4.2 with the command rmfield, see Listing 27.

Listing 27: *removing an attribute from an XML object*

```
>> book = xml(fullfile(pwd,'examples','book.xml'));
>> lang = book.title.('@lang')

lang =

    'en'

>> rmfield(book,'title.@lang');
>> lang = book.title.('@lang')

lang =

    Empty cell array: 0-by-1
```

4.8   Using namespaces

Since element names in XML are not predefined, a name conflict can occur when the same element name is used to describe different content and definitions. For example the element name <table> to describe a piece of furniture and to describe a collection of data. This conflict can be solved by using namespace prefixes to distinguish different types of elements with the same name. This namespace prefix is placed as an attribute in the start tag of an element and has the following syntax:

Listing 28: *namespace definition with the xmlns attribute*

```
xmlns:namespace-prefix="namespaceURI"
```

All child elements with the same prefix are then associated with the same namespace. The only purpose of the string ("namespaceURI") is to identify the namespace with a unique name and often companies use the namespace as a pointer to a real Web page containing information about the namespace. In Listing 29 two namespace prefixes are defined, 'ns' and 'nsdim', with definitions 'http://www.w3schools.com/furniture' and 'http://www.modelit.nl/dimension' respectively.

Listing 29: *An XML document with namespaces*

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<table xmlns:ns="http://www.w3schools.com/furniture"
xmlns:nsdim="http://www.modelit.nl/dimension">
   <ns:name>African Coffee Table</ns:name>
   <width nsdim:dim="centimeter">80</width>
   <length nsdim:dim="meter">1.20</length>
</table>
```

The XML document of Listing 29 can be parsed by the xml command by setting the second argument (isNameSpaceWare) to true, see Listing 30.

Listing 30: *Parsing an XML document with namespaces*

```
>> obj = xml(fullfile(pwd,'examples','namespaces.xml'),1);
```

In order to retrieve the value of the 'name' element (which is in the namespace defined by the prefix 'ns') the XML-object needs to know what the namespace definition of 'ns' is, for example in the XML document of Listing 29 the namespace prefix 'ns' means 'http://www.w3schools.com/furniture'. There are five functions available in the XML toolbox to handle namespace definitions:

- ***addns*** (see section 5.2.15)
  add a namespace definition to the XML object's collection of namespace definitions. In Listing 31 for example the prefix 'ns' is given the definition 'http://www.w3schools.com/furniture' and the prefix 'nsdim' is given the definition 'http://www.modelit.nl/dimension'.

  Listing 31: *add namespace definitions to the XML object*

  ```
  >> obj = addns(obj,{'ns','http://www.w3schools.com/furniture'});
  >> obj = addns(obj,{'nsdim','http://www.modelit.nl/dimension'});
  ```

- ***listns*** (see section 5.2.16)
  display the defined namespaces for the XML object, in Listing 32 the namespace definitions which were added in the previous step are listed.

  Listing 32: *display the defined namespaces in an XML object*

  ```
  >> listns(obj)
  nsdim --> http://www.modelit.nl/dimension
  ns    --> http://www.w3schools.com/furniture
  ```

- ***getns*** (see section 5.2.19)
  retrieve the namespace definition of a namespace prefix, in Listing 33 the namespace definition for the prefix 'ns' is retrieved.

  Listing 33: *retrieval of the namespace definition of the 'ns' prefix*

  ```
  >> getns(obj,'ns')

  ans =

  http://www.w3schools.com/furniture
  ```

- ***removens*** (see section 5.2.18)
  remove a namespace definition from the XML's collection of namespace definitions. In Listing 34 the namespace definition for the prefix 'ns' is removed.

Listing 34: *remove the namespace definition ' ns'*

```
>> obj = removens(obj,'ns')
```

- ***clearns*** (see section 5.2.17)
  remove all namespace definitions from the XML object's collection of namespacedefinitions, see Listing 35.

Listing 35: *remove all namespace definitions*

```
obj = clearns(obj)
```

Now the XML contents can be accessed and manipulated much the same way as described in section 4.2 just by adding the defined namespace prefix and colon to the fieldnames, see Listing 36.

Listing 36: *Accessing an XML document with namespaces*

```
>> obj = xml(fullfile(pwd,'examples','namespaces.xml'),1)

    xml-object (root: table)
    number of nodes: 4
    fieldnames:
    - length  (1)
    - ns:name (1)
    - width   (1)

>> addns(obj,{'ns','http://www.w3schools.com/furniture'})
>> addns(obj,{'nsdim','http://www.modelit.nl/dimension'})
>> listns(obj)
nsdim --> http://www.modelit.nl/dimension
ns    --> http://www.w3schools.com/furniture
>> name = obj.('ns:name')

name =

    'African Coffee Table'

>> obj.length.('@nsdim:dim')

ans =

    'meter'
```

In some cases an XML document contains so-called default namespaces, see Listing 37 for an example.

Listing 37: *An XML document with default namespaces*

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<table xmlns="http://www.w3schools.com/furniture">
   <name>African Coffee Table</name>
   <width>80</width>
   <length>1.20</length>
</table>
```

A namespace is defined as an xmlns attribute in an element, but without a prefix. All child nodes will be automatically associated with that default namespace and takes away the node to use prefixes in all the child elements.

Listing 38: *definition of a default namespace*

```
xmlns="namespaceURI"
```

The contents of an XML document with default namespaces can be accessed and manipulated by adding a dummy namespace prefix and using this prefix in the fieldname as the XML object needs to know in what namespace it should look. Accessing and manipulating the XML contents is then the same as described before.

Listing 39: *Accessing contents in an XML document with default namespaces* **I**

```
>> obj = xml(fullfile(pwd,'examples','default_namespace.xml'));
>> obj = addns(obj,{'ns','http://www.w3schools.com/furniture'});
>> obj.('ns:name')

ans =

    'African Coffee Table'
```

Another possibility to access and manipulate XML contents in document with default namespaces is to construct the XML object with the second argument (isNameSpaceWare) of the constructor set to false and then follow the same steps as described in section 4.2, thus without using any namespace prefixes, see Listing 40.

Listing 40: *Accessing contents in an XML document with default namespaces* **II**

```
>> obj = xml(fullfile(pwd,'examples','default_namespace.xml'),0)

    xml-object (root: table)
    number of nodes: 4
    fieldnames:
    - length (1)
    - name   (1)
    - width  (1)
>> obj.name

ans =

    'African Coffee Table'
```

4.9   Using XPath

XPath is a syntax for navigating through the elements and attributes inside XML documents in order to extract specific information. XPath uses path expressions to select nodes or node-sets in an XML document. These path expressions are similar to the expressions which are used to access the content in Matlab structures.

In Table 1 a short overview of XPath symbols is given. This overview is only a small subset of all the possibilities of XPath in the XML toolbox. The XML object is constructed in such way that the complete set of XPath functionalities can be used. See http://www.w3schools.com/xpath for more information about how to use XPath.

Table 1: *XPath symbols*

XPath symbol	Description	Example
nodename	Select all child nodes of the node	Listing 41
/	Select nodes in the document that match the expression	Listing 41
//	Select nodes in the document that match the expression no matter where they are	Listing 47
*	Wildcard	Listing 46
[ ]	Predicate, to specify a condition which has to be met for a node to be selected	Listing 43 Listing 44 Listing 45
@	Select attributes	Listing 46
position(), last()	Node index	Listing 42

Listing 41: *Select all books with an XPath expression*

```
>> bookstore = xml(fullfile(pwd,'examples','books.xml'))
>> books = xpath(obj,'/bookstore/book')

books =

    [1x1 xml]
    [1x1 xml]
    [1x1 xml]
    [1x1 xml]
```

Listing 42: *Select only the title of the first and last book with an XPath expression*

```
>> bookstore = xml(fullfile(pwd,'examples','books.xml'))
>> titles = xpath(obj,'/bookstore/book[position() = 1 | position() = last()]/title')

titles =

    'Everyday Italian'
    'Learning XML'
```

Listing 43: *Select only the book with title 'Harry Potter' with an XPath expression*

```
>> bookstore = xml(fullfile(pwd,'examples','books.xml'))
>> book = xpath(obj,'/bookstore/book[title=''Harry Potter'']')

book =

    [1x1 xml]
```

Listing 44: *Select only the books which are over 35 euro with an XPath expression*

```
>> bookstore = xml(fullfile(pwd,'examples','books.xml'))
>> books = xpath(bookstore,'/bookstore/book[price>35]')

books =

    [1x1 xml]
    [1x1 xml]
```

Listing 45: *Select the titles of the books over 35 euro with an XPath expression*

```
>> bookstore = xml(fullfile(pwd,'examples','books.xml'))
>> titles = xpath(bookstore,'/bookstore/book[price>35]/title')

titles =

    'XQuery Kick Start'
    'Learning XML'
```

Listing 46: *Select all attributes with an XPath expression*

```
>> bookstore = xml(fullfile(pwd,'examples','books.xml'))
>> xpath(books,'//@*')

    xml-object (root: root)
    number of nodes: 5
    fieldnames:
    - author (1)
    - price  (1)
    - title  (1)
    - year   (1)
```

Listing 47: *Select all authors of all books with an XPath expression*

```
>> bookstore = xml(fullfile(pwd,'examples','books.xml'))
>> authors = xpath(bookstore,'//author')

authors =

    'Giada De Laurentiis'
    'J K. Rowling'
    'James McGovern'
    'Per Bothner'
    'Kurt Cagle'
    'James Linn'
    'Vaidyanathan Nagarajan'
    'Erik T. Ray'
```

It is also possible to change the XML contents in the XML object by using an XPath expression, the only restriction is that no new fields can be created. The XPath expression must thus return one or more elements of the XML document. Listing 48 is a good example of how XPath expression can be used to manipulate the XML contents of an XML object.

Listing 48: *Change the prices of the books 35 euro to 40 euro*

```
>> books = xml(fullfile(pwd,'examples','books.xml'));
>> books = xpath(books,'/bookstore/book[price>35]/price',40);
>> books.book.price

ans =

    '30.00'
    '29.99'
    '40'
    '40'
```

The function isfield (see section 5.2.7) and rmfield (see section 5.2.9) can also be used in combination with XPath expressions, see for example Listing 49.

Listing 49: *Remove the books which are over 35 euro*

```
>> bookstore = xml(fullfile(pwd,'examples','books.xml'));
>> isfield(bookstore,'/bookstore/book[price>35]')

ans =

     1

>> rmfield(bookstore,'/bookstore/book[price>35]');
>> isfield(bookstore,'/bookstore/book[price>35]')

ans =

     0
```

5   Function references

5.1   Constructor

5.1.1   xml

Listing 50: *xml*

```
xml - constructor for an xml-object

  CALL:
   obj = xml(FileName,isNameSpaceAware,isValidating)

  INPUT:
   FileName:          <string> name of the sourcefile
                      <string> the xml string
                      <java-object> with a D(ocument) (O)bject (M)odel
                      <struct> a Matlab structure
   isNameSpaceAware: <boolean> (optional) (default == 1) ignore namespaces
   isValidating:     <boolean> (optional) (default == 0) validate document

  OUTPUT:
   obj: <xml-object> with fields:
                      - DOM:  <java object> the DOM object
                      - file: <string> the name of the xml source
                      - NS:   <java object> a hashmap with namespace
                                            definitions
                      N.B. obj is empty when an error occurred

  See also: xml/view, xml/inspect
```

5.2   Methods of the xml-object

5.2.1   view (has been superseeded by xml2str)

Listing 51: *view*

```
  view - convert the xml-object into a string

  CALL:
   view(obj)

  INPUT:
   obj:      <xml-object>

  OUTPUT:
   S:        <string> with the xml-document

  EXAMPLE:
   %create an xml from a sourcefile
   obj = xml(fullfile(pwd,'examples','books.xml'))
   view(obj)

  See also: xml, xml/save, xml/inspect
```

5.2.2 xml2str

Listing 52: *view*

```
xml2str - convert the xml-object into a string

Note:  This method replaces the method "view"
       the method view is kept for backward compability

CALL:
 xml2str(obj)
 S=xml2str(obj)

INPUT:
 obj:     <xml-object>

OUTPUT:
 S:       <string> with the xml-document

EXAMPLE:
 %create an xml from a sourcefile
 obj = xml(fullfile(pwd,'examples','books.xml'))
 xml2str(obj)

See also: xml, xml/save, xml/inspect
```

5.2.3 inspect

Listing 53: *inspect*

```
inspect - visualize the xml document as a tree in a separate window

CALL:
 inspect(obj)

INPUT:
 obj: <xml-object>

OUTPUT:
 none, the DOM representation of the xml document appears as a tree
       in a separate window

See also: xml, xml/view
```

5.2.4 display

Listing 54: *display*

```
display - display information about an xml-object on the console

CALL:
 display(obj)

INPUT:
 obj: <xml-object>

OUTPUT:
 none, information about the xml-object is displayed on the console

See also: xml, display
```

5.2.5 save

Listing 55: *save*

```
 save - save the xml-object as an xml file

 CALL:
  obj = save(obj,fname)

 INPUT:
  obj:   <xml-object>
  fname: <string> (optional) the name of the xml file, if fname is not
                             specified a save dialog will pop up

 OUTPUT:
  obj: <xml-object> the file field of the xml-object is updated and an xml
                    file is created

 See also: xml, xml/view, xml/inspect
```

5.2.6 fieldnames

Listing 56: *fieldnames*

```
fieldNames - get the names of the direct children of the root node
             c.f. the function fieldnames for structures
  CALL:
   fields = fieldnames(obj)

  INPUT:
   obj: <xml-object>

  OUTPUT:
   fields: <cellstring> with the nodenames of the children of the root node

  See also: xml, xml/getRoot, xml/noNodes, xml/isfield
```

5.2.7 isfield

Listing 57: *isfield*

```
isfield - true if at least one node satisfies the indexing 'sub'

  CALL:
   tf = isfield(obj,field)

  INPUT:
   obj: <xml-object>
   sub: <string> index into xml document (same format as indexing into
                 Matlab structures) e.g. 'book(1)' or 'book(1).title'
                 result in the same substructs as would be obtained if
                 S.book(1)or S.book(1).title were used (S a Matlab
                 structure)
        <string> with xpath expression

  OUTPUT:
   tf: <boolean>  true if at least one node satisfies the indexing 'sub'

  See also: xml, xml/fieldNames, xml/rmfield
```

31

5.2.8 isempty

Listing 58: *isempty*

```
isempty - true if the xml-object has no fields

  CALL:
   tf = isempty(obj)

  INPUT:
   obj: <xml-object>

  OUTPUT:
   tf: <boolean> true if the DOM representation of the xml document does
                 not contain any nodes, or equivalently the xml-document
                 has no fields

  See also: xml, xml/noNodes, xml/fieldnames, xml/getRoot
```

5.2.9 rmfield

Listing 59: *rmfield*

```
rmfield - remove elements and attributes from an xml-object which satisfy
          the indexing 'sub'

  CALL:
   rmfield(obj,sub)

  INPUT:
   obj: <xml-object>
   sub: <string> index into xml document (same format as indexing into
                 Matlab structures) e.g. 'book(1)' or 'book(1).title'
                 result in the same substructs as would be obtained if
                 S.book(1)or S.book(1).title were used (S a Matlab
                 structure)
        <string> with xpath expression

  OUTPUT:
   none, the xml-object is updated

  See also: xml, xml/fieldNames, xml/isfield
```

5.2.10 xpath

Listing 60: *xpath*

```
xpath - carry out a set or get for an xml-object using xpath syntax

CALL:
 S = xpath(obj,ind,data)

INPUT:
 obj:  <xml-object>
 ind:  <struct array> with fields
                     - type: one of '.' or '()'
                     - subs: subscript values (field name or cell array
                             of index vectors)
       <string> with an xpath expression
 data: (optional) with the values to be put in the by ind defined
                  fields in the xml-object, allowed types:
                   - <struct> matlab structure
                   - <xml-object>
           <org.apache.xerces.dom.ElementNSImpl>
           <org.apache.xerces.dom.ElementImpl>

OUTPUT:
 S:       <cell array> in nargin == 2 (get is used)
          <xml-object> if nargin == 3 (set is used)

See also: xml, xml/set, xml/get, xml/subsref, xml/subsasgn,
          xml/private/buildXpath
```

5.2.11 subsasgn

Listing 61: *subsasgn*

```
subsasgn - assign new values to the xml document in an xml-object

CALL:
 obj = subsassgn(obj,ind,data)

INPUT:
 obj:  <xml-object>
 ind:  <struct array> with fields
                     - type: one of '.' or '()'
                     - subs: subscript values (field name or cell array
                             of index vectors)
       <string> with an xpath expression
 data: (optional) with the values to be put in the by ind defined
                  fields in the xml-object, allowed types:
                   - <struct> matlab structure
                   - <xml-object>
                   - <org.apache.xerces.dom.ElementImpl>

OUTPUT:
 obj: <xml-object>

See also: xml, xml/subsref, xml/xpath, subsasgn
```

5.2.12 subsref

Listing 62: *subsref*

```
subsref - subscripted reference for an xml object

CALL:
 S = subsref(obj,ind)

INPUT:
 obj:  <xml-object>
 ind:  <struct array> with fields
                   - type: one of '.' or '()'
                   - subs: subscript values (field name or cell array
                           of index vectors)
       <string> with an xpath expression

OUTPUT:
 S: <cell array> with contents of the referenced nodes, can contain
               xml objects, strings or numbers

See also: xml, xml/subsasgn, xml/xpath, subsref
```

5.2.13 selectNodes

Listing 63: *selectNodes*

```
selectNodes - select nodes from the XML DOM tree

CALL:
 nodesList = selectNodes(obj,ind)

INPUT:
 obj: <xml-object>
 ind: <struct array> with fields
                   - type: one of '.' or '()'
                   - subs: subscript values (field name or cell array
                           of index vectors)
       <string> with an xpath expression

OUTPUT:
 nodesList: <java object> java.util.ArrayList with tree nodes

See also: xml, xml/xpath, xml/subsref, xml/subsasgn,
          xml/private/buildXpath
```

5.2.14 xslt

Listing 64: *xslt*

```
xslt - transform the xml-object to html by using a stylesheet

CALL:
 HTMLstring = xslt(obj,xsl,fileName)

INPUT:
 obj:     <xml-object>
 xsl:     <string> filename of the stylesheet
 fileName: <string> (optional) the name of the file waarnaar
                     de HTML string weggeschreven moet worden.

OUTPUT:
 HTMLstring:  <string> in HTML de getransformeerde XML string

See also: xml, xml/save, web, xslt
```

5.2.15 addns

Listing 65: *addns*

```
addns - add a namespace definition to the xml-object

CALL:
 obj = addns(obj,S)

INPUT:
 obj: <xml-object>
 S:   <struct> fieldnames --> namespace variable
               values      --> namespace value
      <cell array> nx2, first column  --> namespace variable
                        second column --> namespace value

OUTPUT:
 obj: <xml-object>

See also: xml, xml/listns, xml/clearns, xml/removens, xml/getns
```

5.2.16 listns

Listing 66: *listns*

```
listns - list the namespace definitions of the xml-object

CALL:
 listns(obj)

INPUT:
 obj: <xml-object>

OUTPUT:
 no direct output, the defined namespaces are displayed on the console

See also: xml, xml/addns, xml/clearns, xml/removens, xml/getns
```

5.2.17 clearns

Listing 67: *clearns*

```
clearns - remove all the namespace definitions from the xml-object

 CALL:
 obj = addns(obj,S)

 INPUT:
 obj: <xml-object>
 S:   <struct> fieldnames --> namespace variable
               values      --> namespace value
      <cell array> size: nx2, first column  --> namespace variable
                        second column --> namespace value

 OUTPUT:
 obj: <xml-object> with no namespace definitions

 See also: xml, xml/listns, xml/addns, xml/removens, xml/getns
```

5.2.18 removens

Listing 68: *removens*

```
removens - remove a namespace definition from the xml-object

  CALL:
   obj = removens(obj,S)

  INPUT:
   obj: <xml-object>
   S:   <char array> with names of the namespace definitions to be removed
        <cell array> with names of the namespace definitions to be removed

  OUTPUT:
   obj: <xml-object>

  See also: xml, xml/listns, xml/clearns, xml/addns, xml/getns
```

5.2.19 getns

Listing 69: *getns*

```
getns - retrieve a namespace definition from the xml-object

  CALL:
   S = listns(obj,key)

  INPUT:
   obj: <xml-object>
   key: <string> with a namespace variable for which the definition has to
                 be retrieved

  OUTPUT:
   S: <string> with the namespace definition

  See also: xml, xml/addns, xml/clearns, xml/removens, xml/listns
```

5.2.20 get

Listing 70: *get*

```
  get - get the value of the specified property for an xml-object (from the
        object itself not from the xml)

  CALL:
   prop_val = get(obj,prop_name)

  INPUT:
   obj:         <xml-object>
   prop_name:   <string> propertyname, possible values:
                        - DOM  <org.apache.xerces.dom.DeferredDocumentImpl>
                               with the DOM representation of the xml
                        - file <string> with filename
                        - NS   <java.util.HashMap> with namespaces

  OUTPUT:
   prop_val:    the value of the specified property for the xml-object
                <struct> with all properties plus values if nargin == 1

  See also: xml, xml/set
```

5.2.21 set

Listing 71: *set*

```
set - set the value of the specified property for an xml-object

  CALL:
   set(obj,prop_name,prop_value)

  INPUT:
   obj:         <xml-object>
   prop_name:    <string> propertyname, possible values:
                         - DOM  <org.apache.xerces.dom.DeferredDocumentImpl>
                                with the DOM representation of the xml
                         - file <string> with filename
                         - NS   <java.util.HashMap> with namespaces
   prop_value:   the value of the property to be set for the xml-object

  OUTPUT:
   obj: <xml-object> with the property prop_name set to prop_value

  See also: xml, xml/get
```

5.2.22 getRoot

Listing 72: *getRoot*

```
getRoot - get the root node of an xml-object and its name

  CALL:
   [rootname root] = getRoot(obj)

  INPUT:
   obj: <xml-object>

  OUTPUT:
   rootname: <string> the name of the root node
   root:     <java object> org.apache.xerces.dom.DeferredElementNSImpl or
                           org.apache.xerces.dom.DeferredElementImpl

  See also: xml, xml/noNodes
```

5.2.23 noNodes

Listing 73: *noNodes*

```
  noNodes - get the total number of nodes present in the DOM-representation
            of the xml document

  CALL:
   N = noNodes(obj)

  INPUT:
   obj: <xml-object>

  OUTPUT:
   N: <integer> with the total number of nodes in the DOM object

  See also: xml, xml/getRoot
```

5.3 General functions

5.3.1 install

Listing 74: *install*

```
install - add necessary paths for the xml-toolbox to the matlabpath and
          classpath.txt
```

5.3.2 serializeDOM

Listing 75: *serializeDOM*

```
serializeDOM - serialise a DOM by transformation to a string of file

CALL:
 String = serialize(DOM,fileName)

INPUT:
 DOM:       <java-object> org.apache.xerces.dom.DocumentImpl
 fileName: <string> (optional) met een geldige bestandsnaam

OUTPUT:
 String: <string>  if nargin == 1 the serialised DOM
         <boolean> if nargin == 2,
                                   0 -> saving to fileName not successful
                                   1 -> saving to fileName successful

See also: xml, xml/inspect, xml/view, xml/save
```

5.3.3 startup

Listing 76: *startup*

```
startup - startup script for any project using Modelit xml toolbox

NOTES
    merge this file with existing startup.m file if necessary
```

5.3.4 struct2xmlstring

Listing 77: *struct2xmlstring*

```
struct2xmlstr - Fast and simple way to create xml strings from Matlab
                structure. Equivalent to xml2str(xml(S)), but faster.

 CALL
     str=struct2xmlstr(S)
     str=struct2xmlstr(S,rootname)
     str=struct2xmlstr(S,rootname,FLOATFORMAT)
     str=struct2xmlstr(S,'',FLOATFORMAT)

 INPUT
     S
         Matlab structure.
     rootname
         parameter tag to be included as root
     FLOATFORMAT
         formatstring to be used by sprintf to convert non-integer numeric
         parameters to character array. Default value: '%f'. Examples of
         other values: '%.10g';'%.8f'. Any digits beyond specified
         precision will be lost.
```

```
OUTPUT
    str (1xN char array)
        Matlab char array that contains xml representation of S

NOTES
    This is function is useful for conversion of simple Matlab structures
    to XML strings. However it supports only a limited number of
    properties.

EXAMPLES
    struct2xmlstr(S)
    struct2xmlstr(S,'','%.10g')
    struct2xmlstr(S,'','%.8f')
```

5.3.5 xmlpath

Listing 78: *xmlpath*

```
xmlpath - return the path of the xml toolbox

  INPUT
      this function accepts no input arguments

  OUTPUT
      pname
          Matlab char array containg fuill path to root of Modelit xml toolbox.
```

5.3.6 xmlUnitTest

Listing 79: *xmlUnitTest*

```
xmlUnitTest - simple verification of module xml.m with various example
              xml files.
```

39

5.4 Private methods of the xml-object.

5.4.1 buildXPath

Listing 80: *buildXPath*

```
buildXPath - create an XPath object for an XML DOMtree

CALL:
 x = buildXPath(string,nsStruct)

INPUT:
 string:     <string> XPath expression
 Namespaces: <java object> (optional) a java.util.HashMap with namespace
                                      definitions

OUTPUT:
 x: <java object> org.jaxen.dom.DOMXPath

See also: xml, xml/xpath, xml/subsasgn, xml/subsref
```

5.4.2 struct2hash

Listing 81: *struct2hash*

```
struct2hash - convert a matlab structure into a java hashmap

CALL:
 H = struct2hash(S,H)

INPUT:
 S: <struct> fieldnames --> hashmap keys
             values     --> hashmap entries
 H: <java object> (optional) java.util.HashMap

OUTPUT:
 H: <java object> java.util.HashMap

See also: xml, xml/private/buildXpath
```

5.4.3 ind2xpath

Listing 82: *ind2xpath*

```
ind2xpath - convert a matlab substruct into an xpath string

  CALL:
   xpathstr = ind2xpath(ind)

  INPUT:
   ind: <struct> see substruct

  OUTPUT:
   xpathstr: <string> xpath equivalent of the substruct

  See also: xml, xml/private/buildXpath, xml/subsasgn, xml/subsref,
            xml/xpath, substruct
```

5.4.4 emptyDocument

Listing 83: *emptyDocument*

```
  emptyDocument - create an empty Document Object Model (DOM) (only the
                  root node is present)

  CALL:
   document = emptyDocument(root)

  INPUT:
   root: (optional) <string> name of the root node
                    <org.apache.xerces.dom.ElementImpl>
                    <org.apache.xerces.dom.ElementNSImpl>
                    default == 'root'

  OUTPUT:
   document: <java-object> org.apache.xerces.dom.DocumentImpl

  See also: xml, xml/view, xml/inspect
```

5.4.5   sub2ind

Listing 84: *sub2ind*

```
sub2ind - convert a string into a struct array of type substruct, for
          indexing into xml documents as if they were Matlab structures

CALL:
 ind = sub2ind(S)

INPUT:
 S: <string> index into xml document (same format as indexing into
             Matlab structures) e.g. 'book(1)' or 'book(1).title' result
             in the same substructs as would be obtained if S.book(1) or
             S.book(1).title were used (S a Matlab structure).

OUTPUT:
 ind: <struct array> with fields:
                        - type -> subscript types '.', '()', or '{}'
                        - subs -> actual subscript values (field names or
                                  cell arrays of index vectors)

See also: xml, xml/isfield, xml/rmfield, substruct
```

5.4.6   fieldInfo

Listing 85: *fieldInfo*

```
fieldInfo - determine the number and names of the direct children of the
            root node and the name of the root node

CALL:
 S = fieldInfo(obj)

INPUT:
 obj: <xml-object>

OUTPUT:
 S:   <struct> with fields
      - root     : <string> name of the root node
      - children : <struct> with fields
                   - name:     <string> names of the direct children of
                                        the root node
                   - frequency: <int> number of time a certain node
                                       appears

See also: xml, xml/display
```

5.4.7   toString

Listing 86: *toString*

```
toString - convert java object, cellstring or char array to string

CALL:
 S = toString(S)

INPUT:
 S: <cell string>
    <char array>
    <java object>

OUTPUT:
 S: <string>

See also: xml, xml/xpath, xml/subsasgn
```

5.4.8 chararray2char

Listing 87: *chararray2char*

```
chararray2char - convert char array to string

CALL:
 str = chararray2char(str)

INPUT:
 str: <char array>

OUTPUT:
 str: <string>

See also: xml, xml/xpath, xml/subsasgn, xml/private/toString
```

6 Examples

In this section the XML files which are used in the examples are listed. These XML files can also be found in the subdirectory 'example' of the directory where the XML toolbox is installed.

6.1 Books

This example was taken from http://www.w3schools.com.

Listing 88: *books.xml*

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<bookstore>
      <book category="COOKING">
             <title lang="en">Everyday Italian</title>
             <author>Giada De Laurentiis</author>
             <year>2005</year>
             <price>30.00</price>
      </book>
      <book category="CHILDREN">
             <title lang="en">Harry Potter</title>
             <author>J K. Rowling</author>
             <year>2005</year>
             <price>29.99</price>
      </book>
      <book category="WEB">
             <title lang="en">XQuery Kick Start</title>
             <author>James McGovern</author>
             <author>Per Bothner</author>
             <author>Kurt Cagle</author>
             <author>James Linn</author>
             <author>Vaidyanathan Nagarajan</author>
             <year>2003</year>
             <price>49.99</price>
      </book>
      <book category="WEB">
             <title lang="en">Learning XML</title>
             <author>Erik T. Ray</author>
             <year>2003</year>
             <price>39.95</price>
      </book>
</bookstore>
```

Listing 89: *book.xml*

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<book category="MATHS">
      <title lang="en">Probability and Random Processes</title>
      <author>G.R. Grimmett</author>
      <author>D.R. Stirzaker</author>
      <year>1992</year>
      <price>39.00</price>
</book>
```

6.2   cd_catalog

This example was taken from http://www.w3schools.com.

Listing 90: *cd_catalog.xml*

```
<?xml version="1.0" encoding="ISO-8859-1>
<CATALOG>
        <CD>
                <TITLE>Empire Burlesque</TITLE>
                <ARTIST>Bob Dylan</ARTIST>
                <COUNTRY>USA</COUNTRY>
                <COMPANY>Columbia</COMPANY>
                <PRICE>10.90</PRICE>
                <YEAR>1985</YEAR>
        </CD>
        <CD>
                <TITLE>Hide your heart</TITLE>
                <ARTIST>Bonnie Tyler</ARTIST>
                <COUNTRY>UK</COUNTRY>
                <COMPANY>CBS Records</COMPANY>
                <PRICE>9.90</PRICE>
                <YEAR>1988</YEAR>
        </CD>
        <CD>
                <TITLE>Greatest Hits</TITLE>
                <ARTIST>Dolly Parton</ARTIST>
                <COUNTRY>USA</COUNTRY>
                <COMPANY>RCA</COMPANY>
                <PRICE>9.90</PRICE>
                <YEAR>1982</YEAR>
        </CD>
        <CD>
                <TITLE>Still got the blues</TITLE>
                <ARTIST>Gary Moore</ARTIST>
                <COUNTRY>UK</COUNTRY>
                <COMPANY>Virgin records</COMPANY>
                <PRICE>10.20</PRICE>
                <YEAR>1990</YEAR>
        </CD>
        <CD>
                <TITLE>Eros</TITLE>
                <ARTIST>Eros Ramazzotti</ARTIST>
                <COUNTRY>EU</COUNTRY>
                <COMPANY>BMG</COMPANY>
                <PRICE>9.90</PRICE>
                <YEAR>1997</YEAR>
        </CD>
        <CD>
                <TITLE>One night only</TITLE>
                <ARTIST>Bee Gees</ARTIST>
                <COUNTRY>UK</COUNTRY>
                <COMPANY>Polydor</COMPANY>
                <PRICE>10.90</PRICE>
                <YEAR>1998</YEAR>
        </CD>
        <CD>
                <TITLE>Sylvias Mother</TITLE>
                <ARTIST>Dr.Hook</ARTIST>
                <COUNTRY>UK</COUNTRY>
                <COMPANY>CBS</COMPANY>
                <PRICE>8.10</PRICE>
                <YEAR>1973</YEAR>
        </CD>
        <CD>
                <TITLE>Maggie May</TITLE>
                <ARTIST>Rod Stewart</ARTIST>
                <COUNTRY>UK</COUNTRY>
                <COMPANY>Pickwick</COMPANY>
                <PRICE>8.50</PRICE>
                <YEAR>1990</YEAR>
        </CD>
```

```
<CD>
        <TITLE>Romanza</TITLE>
        <ARTIST>Andrea Bocelli</ARTIST>
        <COUNTRY>EU</COUNTRY>
        <COMPANY>Polydor</COMPANY>
        <PRICE>10.80</PRICE>
        <YEAR>1996</YEAR>
</CD>
<CD>
        <TITLE>When a man loves a woman</TITLE>
        <ARTIST>Percy Sledge</ARTIST>
        <COUNTRY>USA</COUNTRY>
        <COMPANY>Atlantic</COMPANY>
        <PRICE>8.70</PRICE>
        <YEAR>1987</YEAR>
</CD>
<CD>
        <TITLE>Black angel</TITLE>
        <ARTIST>Savage Rose</ARTIST>
        <COUNTRY>EU</COUNTRY>
        <COMPANY>Mega</COMPANY>
        <PRICE>10.90</PRICE>
        <YEAR>1995</YEAR>
</CD>
<CD>
        <TITLE>1999 Grammy Nominees</TITLE>
        <ARTIST>Many</ARTIST>
        <COUNTRY>USA</COUNTRY>
        <COMPANY>Grammy</COMPANY>
        <PRICE>10.20</PRICE>
        <YEAR>1999</YEAR>
</CD>
<CD>
        <TITLE>For the good times</TITLE>
        <ARTIST>Kenny Rogers</ARTIST>
        <COUNTRY>UK</COUNTRY>
        <COMPANY>Mucik Master</COMPANY>
        <PRICE>8.70</PRICE>
        <YEAR>1995</YEAR>
</CD>
<CD>
        <TITLE>Big Willie style</TITLE>
        <ARTIST>Will Smith</ARTIST>
        <COUNTRY>USA</COUNTRY>
        <COMPANY>Columbia</COMPANY>
        <PRICE>9.90</PRICE>
        <YEAR>1997</YEAR>
</CD>
<CD>
        <TITLE>Tupelo Honey</TITLE>
        <ARTIST>Van Morrison</ARTIST>
        <COUNTRY>UK</COUNTRY>
        <COMPANY>Polydor</COMPANY>
        <PRICE>8.20</PRICE>
        <YEAR>1971</YEAR>
</CD>
<CD>
        <TITLE>Soulsville</TITLE>
        <ARTIST>Jorn Hoel</ARTIST>
        <COUNTRY>Norway</COUNTRY>
        <COMPANY>WEA</COMPANY>
        <PRICE>7.90</PRICE>
        <YEAR>1996</YEAR>
</CD>
<CD>
        <TITLE>The very best of</TITLE>
        <ARTIST>Cat Stevens</ARTIST>
        <COUNTRY>UK</COUNTRY>
        <COMPANY>Island</COMPANY>
        <PRICE>8.90</PRICE>
        <YEAR>1990</YEAR>
</CD>
<CD>
        <TITLE>Stop</TITLE>
        <ARTIST>Sam Brown</ARTIST>
        <COUNTRY>UK</COUNTRY>
        <COMPANY>A and M</COMPANY>
```

```
                        <PRICE>8.90</PRICE>
                        <YEAR>1988</YEAR>
                </CD>
                <CD>

                        <TITLE>Bridge of Spies</TITLE>
                        <ARTIST>T'Pau</ARTIST>
                        <COUNTRY>UK</COUNTRY>
                        <COMPANY>Siren</COMPANY>
                        <PRICE>7.90</PRICE>
                        <YEAR>1987</YEAR>
                </CD>
                <CD>

                        <TITLE>Private Dancer</TITLE>
                        <ARTIST>Tina Turner</ARTIST>
                        <COUNTRY>UK</COUNTRY>
                        <COMPANY>Capitol</COMPANY>
                        <PRICE>8.90</PRICE>
                        <YEAR>1983</YEAR>
                </CD>
                <CD>

                        <TITLE>Midt om natten</TITLE>
                        <ARTIST>Kim Larsen</ARTIST>
                        <COUNTRY>EU</COUNTRY>
                        <COMPANY>Medley</COMPANY>
                        <PRICE>7.80</PRICE>
                        <YEAR>1983</YEAR>
                </CD>
                <CD>

                        <TITLE>Pavarotti Gala Concert</TITLE>
                        <ARTIST>Luciano Pavarotti</ARTIST>
                        <COUNTRY>UK</COUNTRY>
                        <COMPANY>DECCA</COMPANY>
                        <PRICE>9.90</PRICE>
                        <YEAR>1991</YEAR>
                </CD>
                <CD>
                        <TITLE>The dock of the bay</TITLE>
                        <ARTIST>Otis Redding</ARTIST>
                        <COUNTRY>USA</COUNTRY>
                        <COMPANY>Atlantic</COMPANY>
                        <PRICE>7.90</PRICE>
                        <YEAR>1987</YEAR>
                </CD>
                <CD>
                        <TITLE>Picture book</TITLE>
                        <ARTIST>Simply Red</ARTIST>
                        <COUNTRY>EU</COUNTRY>
                        <COMPANY>Elektra</COMPANY>
                        <PRICE>7.20</PRICE>
                        <YEAR>1985</YEAR>
                </CD>
                <CD>
                        <TITLE>Red</TITLE>
                        <ARTIST>The Communards</ARTIST>
                        <COUNTRY>UK</COUNTRY>
                        <COMPANY>London</COMPANY>
                        <PRICE>7.80</PRICE>
                        <YEAR>1987</YEAR>
                </CD>
                <CD>
                        <TITLE>Unchain my heart</TITLE>
                        <ARTIST>Joe Cocker</ARTIST>
                        <COUNTRY>USA</COUNTRY>
                        <COMPANY>EMI</COMPANY>
                        <PRICE>8.20</PRICE>
                        <YEAR>1987</YEAR>
                </CD>
</CATALOG>
```

Listing 91: *cd_catalog.xsl*

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Edited with XML Spy v2006 (http://www.altova.com) -->
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <html>
  <body>
    <h2>My CD Collection</h2>
    <table border="1">
      <tr bgcolor="#9acd32">
        <th>Title</th>
        <th>Artist</th>
      </tr>
      <xsl:for-each select="CATALOG/CD">
      <tr>
        <td><xsl:value-of select="TITLE" /></td>
        <td><xsl:value-of select="ARTIST" /></td>
      </tr>
      </xsl:for-each>
    </table>
  </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

6.3 Business_card

This example was taken from http://www.brics.dk/~amoeller/XML/

Listing 92: *business_card.xml*

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<card type="simple">
   <name>John Doe</name>
   <title>CEO, Widget Inc.</title>
   <email>john.doe@widget.com</email>
   <phone>(202) 456-1414</phone>
 </card>
```

Listing 93: *business_card.xsl*

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0"
                xmlns="http://www.w3.org/1999/xhtml">

   <xsl:template match="card[@type='simple']">
     <html xmlns="http://www.w3.org/1999/xhtml">
       <title>business card</title><body>
         <xsl:apply-templates select="name"/>
         <xsl:apply-templates select="title"/>
         <xsl:apply-templates select="email"/>
         <xsl:apply-templates select="phone"/>
     </body></html>
   </xsl:template>

   <xsl:template match="card/name">
     <h1><xsl:value-of select="text()"/></h1>
   </xsl:template>

   <xsl:template match="email">
     <p>email: <a href="mailto:{text()}"><tt>
       <xsl:value-of select="text()"/>
     </tt></a></p>
   </xsl:template>

 </xsl:stylesheet>
```

6.4 Note

This example was taken from http://www.w3schools.com

Listing 94: *note_dtd.xml*

```
<?xml version="1.0"?>
<!DOCTYPE note SYSTEM "note.dtd">
<note>
        <to>Tove</to>
        <from>Jani</from>
        <heading>Reminder</heading>
        <body>Don't forget me this weekend!</body>
</note>
```

Listing 95: *note.dtd*

```
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

Listing 96: *note_xsd.xml*

```
<?xml version="1.0"?><note
xmlns="http://www.w3schools.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.w3schools.com note.xsd">
        <to>Tove</to>
        <from>Jani</from>
        <heading>Reminder</heading>
        <body>Don't forget me this weekend!</body>
</note>
```

Listing 97: *note.xsd*

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.w3schools.com"
xmlns="http://www.w3schools.com"
elementFormDefault="qualified"><xs:element name="note">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="to" type="xs:string"/>
        <xs:element name="from" type="xs:string"/>
        <xs:element name="heading" type="xs:string"/>
        <xs:element name="body" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
</xs:element></xs:schema>
```

6.5 Plant_catalog

This example was taken from http://www.w3schools.com

Listing 98: *plant_catalog.xml*

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<CATALOG>
  <PLANT>
    <COMMON>Bloodroot</COMMON>
    <BOTANICAL>Sanguinaria canadensis</BOTANICAL>
    <ZONE>4</ZONE>
    <LIGHT>Mostly Shady</LIGHT>
    <PRICE>$2.44</PRICE>
    <AVAILABILITY>031599</AVAILABILITY>
  </PLANT>

  <PLANT>
    <COMMON>Columbine</COMMON>
    <BOTANICAL>Aquilegia canadensis</BOTANICAL>
    <ZONE>3</ZONE>
    <LIGHT>Mostly Shady</LIGHT>
    <PRICE>$9.37</PRICE>
    <AVAILABILITY>030699</AVAILABILITY>
  </PLANT>

  <PLANT>
    <COMMON>Marsh Marigold</COMMON>
    <BOTANICAL>Caltha palustris</BOTANICAL>
    <ZONE>4</ZONE>
    <LIGHT>Mostly Sunny</LIGHT>
    <PRICE>$6.81</PRICE>
    <AVAILABILITY>051799</AVAILABILITY>
  </PLANT>

  <PLANT>
    <COMMON>Cowslip</COMMON>
    <BOTANICAL>Caltha palustris</BOTANICAL>
    <ZONE>4</ZONE>
    <LIGHT>Mostly Shady</LIGHT>
    <PRICE>$9.90</PRICE>
    <AVAILABILITY>030699</AVAILABILITY>
  </PLANT>

  <PLANT>
    <COMMON>Dutchman's-Breeches</COMMON>
    <BOTANICAL>Diecentra cucullaria</BOTANICAL>
    <ZONE>3</ZONE>
    <LIGHT>Mostly Shady</LIGHT>
    <PRICE>$6.44</PRICE>
    <AVAILABILITY>012099</AVAILABILITY>
  </PLANT>

  <PLANT>
    <COMMON>Ginger, Wild</COMMON>
    <BOTANICAL>Asarum canadense</BOTANICAL>
    <ZONE>3</ZONE>
    <LIGHT>Mostly Shady</LIGHT>
    <PRICE>$9.03</PRICE>
    <AVAILABILITY>041899</AVAILABILITY>
  </PLANT>

  <PLANT>
    <COMMON>Hepatica</COMMON>
    <BOTANICAL>Hepatica americana</BOTANICAL>
    <ZONE>4</ZONE>
    <LIGHT>Mostly Shady</LIGHT>
    <PRICE>$4.45</PRICE>
    <AVAILABILITY>012699</AVAILABILITY>
  </PLANT>

  <PLANT>
```

```
    <COMMON>Liverleaf</COMMON>
    <BOTANICAL>Hepatica americana</BOTANICAL>
    <ZONE>4</ZONE>
    <LIGHT>Mostly Shady</LIGHT>
    <PRICE>$3.99</PRICE>
    <AVAILABILITY>010299</AVAILABILITY>
  </PLANT>

  <PLANT>
    <COMMON>Jack-In-The-Pulpit</COMMON>
    <BOTANICAL>Arisaema triphyllum</BOTANICAL>
    <ZONE>4</ZONE>
    <LIGHT>Mostly Shady</LIGHT>
    <PRICE>$3.23</PRICE>
    <AVAILABILITY>020199</AVAILABILITY>
  </PLANT>

  <PLANT>
    <COMMON>Mayapple</COMMON>
    <BOTANICAL>Podophyllum peltatum</BOTANICAL>
    <ZONE>3</ZONE>
    <LIGHT>Mostly Shady</LIGHT>
    <PRICE>$2.98</PRICE>
    <AVAILABILITY>060599</AVAILABILITY>
  </PLANT>

  <PLANT>
    <COMMON>Phlox, Woodland</COMMON>
    <BOTANICAL>Phlox divaricata</BOTANICAL>
    <ZONE>3</ZONE>
    <LIGHT>Sun or Shade</LIGHT>
    <PRICE>$2.80</PRICE>
    <AVAILABILITY>012299</AVAILABILITY>
  </PLANT>

  <PLANT>
    <COMMON>Phlox, Blue</COMMON>
    <BOTANICAL>Phlox divaricata</BOTANICAL>
    <ZONE>3</ZONE>
    <LIGHT>Sun or Shade</LIGHT>
    <PRICE>$5.59</PRICE>
    <AVAILABILITY>021699</AVAILABILITY>
  </PLANT>

  <PLANT>
    <COMMON>Spring-Beauty</COMMON>
    <BOTANICAL>Claytonia Virginica</BOTANICAL>
    <ZONE>7</ZONE>
    <LIGHT>Mostly Shady</LIGHT>
    <PRICE>$6.59</PRICE>
    <AVAILABILITY>020199</AVAILABILITY>
  </PLANT>

  <PLANT>
    <COMMON>Trillium</COMMON>
    <BOTANICAL>Trillium grandiflorum</BOTANICAL>
    <ZONE>5</ZONE>
    <LIGHT>Sun or Shade</LIGHT>
    <PRICE>$3.90</PRICE>
    <AVAILABILITY>042999</AVAILABILITY>
  </PLANT>

  <PLANT>
    <COMMON>Wake Robin</COMMON>
    <BOTANICAL>Trillium grandiflorum</BOTANICAL>
    <ZONE>5</ZONE>
    <LIGHT>Sun or Shade</LIGHT>
    <PRICE>$3.20</PRICE>
    <AVAILABILITY>022199</AVAILABILITY>
  </PLANT>

  <PLANT>
    <COMMON>Violet, Dog-Tooth</COMMON>
    <BOTANICAL>Erythronium americanum</BOTANICAL>
    <ZONE>4</ZONE>
    <LIGHT>Shade</LIGHT>
    <PRICE>$9.04</PRICE>
```

```
    <AVAILABILITY>020199</AVAILABILITY>
  </PLANT>

  <PLANT>
    <COMMON>Trout Lily</COMMON>
    <BOTANICAL>Erythronium americanum</BOTANICAL>
    <ZONE>4</ZONE>
    <LIGHT>Shade</LIGHT>
    <PRICE>$6.94</PRICE>
    <AVAILABILITY>032499</AVAILABILITY>
  </PLANT>

  <PLANT>
    <COMMON>Adder's-Tongue</COMMON>
    <BOTANICAL>Erythronium americanum</BOTANICAL>
    <ZONE>4</ZONE>
    <LIGHT>Shade</LIGHT>
    <PRICE>$9.58</PRICE>
    <AVAILABILITY>041399</AVAILABILITY>
  </PLANT>

  <PLANT>
    <COMMON>Anemone</COMMON>
    <BOTANICAL>Anemone blanda</BOTANICAL>
    <ZONE>6</ZONE>
    <LIGHT>Mostly Shady</LIGHT>
    <PRICE>$8.86</PRICE>
    <AVAILABILITY>122698</AVAILABILITY>
  </PLANT>

  <PLANT>
    <COMMON>Grecian Windflower</COMMON>
    <BOTANICAL>Anemone blanda</BOTANICAL>
    <ZONE>6</ZONE>
    <LIGHT>Mostly Shady</LIGHT>
    <PRICE>$9.16</PRICE>
    <AVAILABILITY>071099</AVAILABILITY>
  </PLANT>

  <PLANT>
    <COMMON>Bee Balm</COMMON>
    <BOTANICAL>Monarda didyma</BOTANICAL>
    <ZONE>4</ZONE>
    <LIGHT>Shade</LIGHT>
    <PRICE>$4.59</PRICE>
    <AVAILABILITY>050399</AVAILABILITY>
  </PLANT>

  <PLANT>
    <COMMON>Bergamont</COMMON>
    <BOTANICAL>Monarda didyma</BOTANICAL>
    <ZONE>4</ZONE>
    <LIGHT>Shade</LIGHT>
    <PRICE>$7.16</PRICE>
    <AVAILABILITY>042799</AVAILABILITY>
  </PLANT>

  <PLANT>
    <COMMON>Black-Eyed Susan</COMMON>
    <BOTANICAL>Rudbeckia hirta</BOTANICAL>
    <ZONE>Annual</ZONE>
    <LIGHT>Sunny</LIGHT>
    <PRICE>$9.80</PRICE>
    <AVAILABILITY>061899</AVAILABILITY>
  </PLANT>

  <PLANT>
    <COMMON>Buttercup</COMMON>
    <BOTANICAL>Ranunculus</BOTANICAL>
    <ZONE>4</ZONE>
    <LIGHT>Shade</LIGHT>
    <PRICE>$2.57</PRICE>
    <AVAILABILITY>061099</AVAILABILITY>
  </PLANT>

  <PLANT>
    <COMMON>Crowfoot</COMMON>
```

```
    <BOTANICAL>Ranunculus</BOTANICAL>
    <ZONE>4</ZONE>
    <LIGHT>Shade</LIGHT>
    <PRICE>$9.34</PRICE>
    <AVAILABILITY>040399</AVAILABILITY>
</PLANT>

<PLANT>
  <COMMON>Butterfly Weed</COMMON>
  <BOTANICAL>Asclepias tuberosa</BOTANICAL>
  <ZONE>Annual</ZONE>
  <LIGHT>Sunny</LIGHT>
  <PRICE>$2.78</PRICE>
  <AVAILABILITY>063099</AVAILABILITY>
</PLANT>

<PLANT>
  <COMMON>Cinquefoil</COMMON>
  <BOTANICAL>Potentilla</BOTANICAL>
  <ZONE>Annual</ZONE>
  <LIGHT>Shade</LIGHT>
  <PRICE>$7.06</PRICE>
  <AVAILABILITY>052599</AVAILABILITY>
</PLANT>

<PLANT>
  <COMMON>Primrose</COMMON>
  <BOTANICAL>Oenothera</BOTANICAL>
  <ZONE>3 - 5</ZONE>
  <LIGHT>Sunny</LIGHT>
  <PRICE>$6.56</PRICE>
  <AVAILABILITY>013099</AVAILABILITY>
</PLANT>

<PLANT>
  <COMMON>Gentian</COMMON>
  <BOTANICAL>Gentiana</BOTANICAL>
  <ZONE>4</ZONE>
  <LIGHT>Sun or Shade</LIGHT>
  <PRICE>$7.81</PRICE>
  <AVAILABILITY>051899</AVAILABILITY>
</PLANT>

<PLANT>
  <COMMON>Blue Gentian</COMMON>
  <BOTANICAL>Gentiana</BOTANICAL>
  <ZONE>4</ZONE>
  <LIGHT>Sun or Shade</LIGHT>
  <PRICE>$8.56</PRICE>
  <AVAILABILITY>050299</AVAILABILITY>
</PLANT>

<PLANT>
  <COMMON>Jacob's Ladder</COMMON>
  <BOTANICAL>Polemonium caeruleum</BOTANICAL>
  <ZONE>Annual</ZONE>
  <LIGHT>Shade</LIGHT>
  <PRICE>$9.26</PRICE>
  <AVAILABILITY>022199</AVAILABILITY>
</PLANT>

<PLANT>
  <COMMON>Greek Valerian</COMMON>
  <BOTANICAL>Polemonium caeruleum</BOTANICAL>
  <ZONE>Annual</ZONE>
  <LIGHT>Shade</LIGHT>
  <PRICE>$4.36</PRICE>
  <AVAILABILITY>071499</AVAILABILITY>
</PLANT>

<PLANT>
  <COMMON>California Poppy</COMMON>
  <BOTANICAL>Eschscholzia californica</BOTANICAL>
  <ZONE>Annual</ZONE>
  <LIGHT>Sun</LIGHT>
  <PRICE>$7.89</PRICE>
  <AVAILABILITY>032799</AVAILABILITY>
```

```
    </PLANT>

    <PLANT>
      <COMMON>Shooting Star</COMMON>
      <BOTANICAL>Dodecatheon</BOTANICAL>
      <ZONE>Annual</ZONE>
      <LIGHT>Mostly Shady</LIGHT>
      <PRICE>$8.60</PRICE>
      <AVAILABILITY>051399</AVAILABILITY>
    </PLANT>

    <PLANT>
      <COMMON>Snakeroot</COMMON>
      <BOTANICAL>Cimicifuga</BOTANICAL>
      <ZONE>Annual</ZONE>
      <LIGHT>Shade</LIGHT>
      <PRICE>$5.63</PRICE>
      <AVAILABILITY>071199</AVAILABILITY>
    </PLANT>

    <PLANT>
      <COMMON>Cardinal Flower</COMMON>
      <BOTANICAL>Lobelia cardinalis</BOTANICAL>
      <ZONE>2</ZONE>
      <LIGHT>Shade</LIGHT>
      <PRICE>$3.02</PRICE>
      <AVAILABILITY>022299</AVAILABILITY>
    </PLANT>
</CATALOG>
```

6.6 namespaces

This example is adapted from http://www.w3schools.com

Listing 99: *namespaces.xml*

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<table xmlns:ns="http://www.w3schools.com/furniture"
 xmlns:nsdim="http://www.modelit.nl/dimension">
   <ns:name>African Coffee Table</ns:name>
   <width nsdim:dim="centimeter">80</width>
   <length nsdim:dim="meter">1.20</length>
</table>
```

6.7   default_namespace

This example is adapted from http://www.w3schools.com

Listing 100: *default_namespace.xml*

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<table xmlns="http://www.w3schools.com/furniture">
   <name>African Coffee Table</name>
   <width>80</width>
   <length>1.20</length>
</table>
```

6.8   mixed

This example is adapted from http://www.w3schools.com

Listing 101: *mixed.xml*

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<collection>
        <item>
                <TITLE>Empire Burlesque</TITLE>
                <ARTIST>Bob Dylan</ARTIST>
                <COUNTRY>USA</COUNTRY>
                <COMPANY>Columbia</COMPANY>
                <PRICE>10.90</PRICE>
                <YEAR>1985</YEAR>
        </item>
        <item>
                <title lang="en">Harry Potter</title>
                <author>J K. Rowling</author>
                <year>2005</year>
                <price>29.99</price>
        </item>
</collection>
```